

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Построение и анализ алгоритмов»
Тема: редакционное расстояние Левенштейна

Студент гр. 3388

Березовский М.А.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы:

Изучить теоретические основы алгоритма Левенштейна.

Задание:

Реализовать алгоритм Левенштейна (частный случай алгоритма Вагнера-Фишера)

Параметры входных данных:

Первая строка входных данных содержит строку из строчных латинских букв. ($SS, 1 \leq |S| \leq 2550, 1 \leq |S| \leq 2550$).

Вторая строка входных данных содержит строку из строчных латинских букв. ($TT, 1 \leq |T| \leq 2550, 1 \leq |T| \leq 2550$).

Параметры выходных данных:

Одно число LL , равное расстоянию Левенштейна между строками SS и TT .

Sample Input:

pedestal
stien

Sample Output:

7

Выполнение работы

Описание алгоритма для решения задачи

Расстояния Левенштейна - минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Алгоритм выполняется в два этапа:

Изначально на вход поступают две строки – S и T

i — текущий символ строки S

j — текущий символ строки T

1. Инициализация таблицы dp

- Создаётся двумерный массив $dp[n+1][m+1]$, где $n = S.length()$, $m = T.length()$.

- Строка S - первая строка, которую нужно превратить, строка T - цель.

- Для каждой позиции (i, j) таблица будет хранить минимальное количество операций, чтобы превратить подстроку $S[0..i-1]$ в подстроку $T[0..j-1]$.

- Первая строка и первый столбец заполняются числами $0..n$ и $0..m$ соответственно:

- $dp[i][0] = i \rightarrow$ нужно удалить i символов из S, чтобы получить пустую строку.

- $dp[0][j] = j \rightarrow$ нужно вставить j символов, чтобы получить подстроку $T[0..j-1]$.

Таким образом, таблица готова для пошагового заполнения.

2. Заполнение таблицы dp

- Алгоритм проходит по всем символам строк S и T
- На каждой итерации выполняется сравнение символов $S[i-1]$ и $T[j-1]$:

1) Совпадение символов

- Если $S[i-1] == T[j-1]$, значит, для этих позиций операций не нужно.
- Тогда $dp[i][j] = dp[i-1][j-1]$
- Идея: текущее редактирование не увеличивает количество операций, мы просто переносим значение предыдущей диагонали.

2) Несовпадение символов

- Если символы различны, возможны три операции:
 - Замена: $dp[i-1][j-1] + 1$
 - Удаление: $dp[i-1][j] + 1$
 - Вставка: $dp[i][j-1] + 1$
- Выбирается минимальное из трёх значений: $dp[i][j] = \min(\text{replace}, \text{delete}, \text{insert})$
- Таким образом, на каждом шаге выбирается минимальное количество действий, необходимое для приведения подстроки $S[0..i-1]$ к подстроке $T[0..j-1]$.

После заполнения всей таблицы значение $dp[n][m]$ — это минимальное количество операций, чтобы превратить строку S в строку T .

Оценка сложности алгоритма:

Оценка времени выполнения:

Алгоритм Левенштейна строит таблицу dp , где строки соответствуют символам первой строки S , а столбцы - символам второй строки T . Для каждой позиции в этой таблице алгоритм сравнивает символы двух строк и выбирает минимальное число операций: вставка, удаление или замена. Так как всего таких позиций $(n+1) \times (m+1)$, где n - длина первой строки, а m - длина второй, каждая клетка обрабатывается один раз, значит, общее количество операций примерно равно $n * m$ — то есть сложность $O(n * m)$

Оценка использования памяти:

Основное место в памяти занимает двумерная таблица dp для хранения промежуточных значений. Дополнительно используются несколько переменных для хранения текущих символов и вычислений — их количество не зависит от длины строк. Таким образом, память растёт примерно пропорционально количеству клеток таблицы: $(n+1) \times (m+1) - O(n * m)$

Тестирование

Таблица 1. Тестирование.

<i>Входные данные</i>	<i>Выходные данные</i>
abc abc	0
abc abd	1
pedestal stien	7
kitten sitting	3

Вывод

В ходе лабораторной работы был изучен и реализован алгоритм вычисления расстояния Левенштейна между двумя строками. Программа создаёт двумерную таблицу dp , в которой хранится минимальное количество операций (вставка, удаление, замена), необходимых для превращения одной строки в другую. Алгоритм пошагово заполняет таблицу, сравнивая символы строк и выбирая оптимальное действие для каждой позиции. Код был подробно проверен на различных тестовых данных, включая полностью совпадающие строки, строки с заменами, вставками и удалениями символов, а также полностью различные строки. Результаты показали корректность работы программы и позволили визуально убедиться в том, как формируются значения таблицы и как выбираются минимальные действия на каждом шаге.